AI Project: Connect Four[1]

Goal: to get a practical understanding of the minimax algorithm with alpha-beta pruning.

This assignment is about a program that plays Connect Four (check `https://en.wikipedia.org/wiki/Connect_Four` or whatever source you like). I realized that it would not be reasonable within this course to write such a program from scratch, and there are many available on the Internet, so I picked one, with permission of the author. (it appears that it's no longer on the Internet). The code is posted on our elearning platform. The goal of the assignment is to study the code from an AI-perspective. You can learn things about applets, object-oriented design patterns, and writing useful comments from it as well, if you want, but that is not the goal.

1 The game

You must think of the board exactly as it is on your screen, that is, it stands up. There are two players (black and red in this implementation). In a move, they choose a column and drop their token in it. It drops down until it's on top of another token, or the bottom. The player who is first to have 4 tokens in a row (horizontally, vertically, or diagonally) wins. The game ends in a draw, if the board is full and no player has 4-in-a-row. If you have not seen the game before, just play a few games to see how it works.

2 The code

The ZIP-file contains

- The `*.java` files with generic game-playing code in Java
  (note: the files `AllPossibleMovesPlayer.java` and `DefaultGameEventListener` are not used).
- The `C4*.java` files with the Connect Four game-playing code in Java
  (the root file is `C4Applet.java`)
- The images for the board, red piece and black piece
- A `c4.html` to start the game in a browser.

[1]This text was taken verbatim from `http://www.it.uu.se/edu/course/homepage/ai/ht10/AI_assignment_2.html`, an artificial intelligence course website at Uppsala University. We merely changed the layout from HTML, but kept the original wording with minute modifications.

To compile this Java code under Unix, use `javac C4Applet.java`. To run the applet under Unix, use `appletviewer c4.html`. The code is from 1999, and javac complains about a deprecated interface. But it works.

The assignment below suggests some changes in the code. You do not need to worry about knowing Java. If you know programming, then you can read the code, understand those parts that you need to understand, and make the necessary modifications.

I suggest that you first read the questions, and then spend some time finding your way in the code. The most interesting files are `MinimaxPlayer.java` and `C4Board.java`. The computer uses the MinimaxPlayer to determine its next move. The search depth (called "level") can be set in the interface. *In the console window, the total number of investigated moves is counted.* The `C4Board` does most of the "game administration". One peculiar method is `getPossibleMoves`, that is supposed to return the moves (= columns) that are valid. In fact, it always returns all columns, which explains some extra tests in the MinimaxPlayer.

3 Questions

The questions 1–5 ask you to measure the performance of modified versions of the Minimax-Player (in terms of the number of moves considered).

What to hand in:

- **the measurements in a decent table,**
- **the procedures with modified code that are relevant (in each case a few lines)**
- **combined in a 15 minute presentation, introducing the project itself, your approach and the collected results.**

Example of the table:

question:	1	2	3	4	5a	5b
move 1:	1882	4680	1445	1225	4680	591
move 2:						
move 3:						

...

The exact numbers depend on the moves you choose (except 4680, see question 2), but these are roughly the numbers you can expect (in the original version, you get 1882 if you play in column 4, but 2667 if you play in column 5 – that's how roughly it is).

Try to measure all versions on the same sequence of moves (this works for all modifications that do not alter the computer's moves). It is enough to measure the first 10 or so moves. You can try different levels, but level 4 (the default) is already quite informative.

1. Measure the original version.

2. The MinimaxPlayer does some alpha-beta pruning. Identify the code that is responsible, and remove it (comment it out) so that we get a pure minimax version. Observe that there are two similar procedures, one for max-nodes and one for min-nodes. Measure the number of moves considered in the first few steps of the game. *Explain why the outcome has to be exactly this number* (that is, if the outcome is x, give a formula that computes x) .

3. Go back to the original version. The alpha-beta pruning does not prune if the current value is equal to the pruning limit. Change this and measure the effect.

4. The code does not use "deep" alpha-beta pruning. That is, the code prunes based on information from the parent, but not on information collected from higher up in the tree (for instance the great-grandparent, as in the last slide of the powerpoint presentation). Change the code so that it does use "deep" alpha-beta pruning, and measure the effect. Hint: you need to pass both alpha and beta as arguments.

5. The method `getPossibleMoves` determines the order in which the moves are considered, in this case from one side to the other. For minimax without pruning, this order is irrelevant, but it is known that alpha-beta pruning works best if you try the best moves first. Of course, there is no way to know a priori what the best moves are, but a useful *heuristic* in Connect Four is that the columns in the middle are usually better than the ones on the sides. Change the program such that it tries the columns from the middle to both sides (you can hard-code the sequence in an array – no need for general code here. It's the measurement that is interesting). Measure the effect in combination with minimax without pruning (a) – there should be no effect, and with deep alpha-beta pruning (b). (This change will change how the computer plays, as optimal moves with the same heuristic value will be found in different order.)

6. The board has 8 columns. Most versions of Connect Four have only 7 columns. In `C4Board.java`, set the `NUMBER_OF_COLUMNS` to 7. Observe that you can still play the game. Apart from the board picture, there is one other place in the code that is not adapted to a different number of columns: the number 84 that occurs unexplained in `C4Board.java`. What is this number, and what should it be in terms of `NUMBER_OF_COLUMNS` and `NUMBER_OF_ROWS`?

7. Explain the heuristic function that evaluates states at the bottom of the search. (2009 bug report: the program does not correctly handle "restart" if the previous game is not finished. The bug adds a constant to the heuristic function, so it is in fact harmless.)
   (a) What quantities does it compute? How does it weigh them?
   (b) How is it computed? Explain how the different objects involved communicate to produce the result (I admit that this question is outside the goal of the assignment. But too interesting not to ask).

8. Sometimes the computer can win in one move, but does not do so. Can you explain why?

4 What to turn in and grading

Please prepare a 15-20 minute presentation introducing the tasks and explaining your solution and results. Turn in your a presentation slides as well as the code and the documentation of your answers. You will be graded based on the quality, clarity and correctness of you presentation and the handed-in documentation.